A SPARQL Query Transformation Rule Language — Application to Retrieval and Adaptation in Case-Based Reasoning

Olivier Bruneau[1], Emmanuelle Gaillard[2], Nicolas Lasolle[1], Jean Lieber[2], Emmanuel Nauer[2], and Justine Reynaud[2]

[1] University of Lorraine, LHSP-AHP, 91 avenue de la Libération, BP 454, F-54001 Nancy cedex, France, olivier.bruneau@univ-lorraine.fr, nicolas.lasolle@telecomnancy.eu
[2] UL, CNRS, Inria, Loria, F-54000 Nancy, {firstname.lastname}@loria.fr

Abstract. This paper presents SQTRL, a language for transformation rules for SPARQL queries, a tool associated with it, and how it can be applied to retrieval and adaptation in case-based reasoning (CBR). Three applications of SQTRL are presented in the domains of cooking and digital humanities. For a CBR system using RDFS for representing cases and domain knowledge, and SPARQL for its query language, case retrieval with SQTRL consists in a minimal modification of the query so that it matches at least a source case. Adaptation based on the modification of an RDFS base can also be handled with the help of this tool. SQTRL and its tool can therefore be used for several goals related to CBR systems based on the semantic web standards RDFS and SPARQL.

Keywords: RDFS, SPARQL, query transformation, retrieval, adaptation, application

1 Introduction

This paper presents a language and a tool that have proven to be useful for addressing three application problems related to the issues of retrieval and adaptation in case-based reasoning (CBR [19]), when the underlying representation language is the semantic web standard RDFS [3].

CBR aims at solving problems by reusing previously solved problems. It is often considered to be a methodology [21]. Indeed, its principles are independent from a knowledge representation language. The upside of this is that it covers a huge family of problem-solving issues, in many application domains. The downside of it is that the application of CBR to a particular domain, with a given representation language, often requires to reimplement most of the CBR steps. However, many studies have been carried out to fill the gap between general principles and particular applications. Some general CBR shells, like JColibri [18] have been implemented and distributed. Furthermore, some tools have been implemented for particular types of problems (e.g., case-based planning [9]) and/or particular types of formalisms (e.g., workflows [15]).

This paper presents a formalism and a tool of general use for the development of CBR systems based on the representation language RDFS, thus it contributes to filling the theory-application gap in CBR. RDFS can be seen as a language combining attribute-value pairs and the use of hierarchies, two features commonly used in CBR systems [13]. The standard query language of RDFS is SPARQL. The language presented in this work is named SQTRL and is a language of rules to transform queries written in SPARQL.

This paper is organized as follows. Section 2 presents preliminaries on RDFS and SPARQL. Then, three application problems that have motivated this work are presented in Section 3. Section 4 presents SQTRL and a tool to manage rules of this language. A discussion pointing out some related work is given in Section 5. Section 6 concludes and highlights some future work.

2 Preliminaries: RDFS and SPARQL

RDFS (for RDF Schema [3]) is a knowlege representation formalism based on RDF, a resource description framework defined on several syntaxes. In RDF, a *resource* is either an *identified resource* or a *variable* (also called blank node or anonymous resource) that is represented in this paper by an identifier starting with the '`?`' character (e.g. `?x`) and can be interpreted as an existentially quantified variable. Some resources are *properties*; they are intended to represent binary relations. A literal is a value of a simple datatype (e.g., an integer, a string, etc.). An RDF base is a set of *triples* $\langle s \;\; p \;\; o \rangle$, where s—the subject of the triple—is a resource, p—the predicate—is a property, and o—the object—is either a resource or a literal. For example $\mathcal{B} = \{\langle \texttt{tarteTatin ing ?x} \rangle, \;\; \langle \texttt{?x type Apple} \rangle\}$ means that the tarte Tatin has an ingredient of type apple.

Some RDF resources constitute the so-called RDFS vocabulary. For the sake of simplicity, only three such resources are considered in this paper: `rdf:type` (abbreviated in `type`), `rdfs:subClassOf` (abbreviated in `subc`), and `rdfs:subPropertyOf` (abbreviated in `subp`). These resources are properties and have the following meaning

$$\langle a \;\; \texttt{type} \;\; C \rangle \text{ means that } a^{\mathcal{I}} \in C^{\mathcal{I}} \qquad \langle C \;\; \texttt{subc} \;\; D \rangle \text{ means that } C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$$
$$\langle p \;\; \texttt{subp} \;\; q \rangle \text{ means that } p^{\mathcal{I}} \subseteq q^{\mathcal{I}} \text{ i.e., if } (x, y) \in p^{\mathcal{I}} \text{ then } (x, y) \in q^{\mathcal{I}}$$

where $a^{\mathcal{I}}$ is an object represented by a, $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ are sets represented by the classes C and D, and $p^{\mathcal{I}}$ and $q^{\mathcal{I}}$ are relations represented by properties p and q. For example, $\langle \texttt{tarteTatin type DessertRecipe} \rangle$ means that the tarte Tatin is a dessert, $\langle \texttt{Apple subc Fruit} \rangle$ means that an apple is a fruit, and $\langle \texttt{mainIng subp ing} \rangle$ means that the main ingredient of a recipe is an ingredient of this recipe. The inference rules $\dfrac{\langle a \;\; \texttt{type} \;\; C \rangle \quad \langle C \;\; \texttt{subc} \;\; D \rangle}{\langle a \;\; \texttt{type} \;\; D \rangle}$, $\dfrac{\langle C \;\; \texttt{subc} \;\; D \rangle \quad \langle D \;\; \texttt{subc} \;\; E \rangle}{\langle C \;\; \texttt{subc} \;\; E \rangle}$, $\dfrac{\langle p \;\; \texttt{subp} \;\; q \rangle \quad \langle q \;\; \texttt{subp} \;\; r \rangle}{\langle p \;\; \texttt{subp} \;\; r \rangle}$ and $\dfrac{\langle a \;\; p \;\; b \rangle \quad \langle p \;\; \texttt{subp} \;\; q \rangle}{\langle a \;\; q \;\; b \rangle}$ are used to define the inference relation $\vdash$. In this paper, every RDFS base $\mathcal{B}$ is considered up to entailment, meaning that if $\mathcal{B} \vdash \tau$, then the triple τ is considered as an element of $\mathcal{B}$.

```
SELECT ?r                                                    select
WHERE {?r  type  DessertRecipe   .          the recipes of desserts
       ?r  ing  ?x  .                        having an ingredient
       ?x  type  Fruit  .                        of type fruit
       ?r  prepTimeInMinutes  ?t  .      and a preparation time
       FILTER (?t  <=  90)}                  of at most 90 minutes
```

Fig. 1. Example of SPARQL query.

To access an RDFS base, a SPARQL query is used. Figure 1 shows such a query. More generally, a SPARQL query $\mathtt{Q}$ is constituted by a "SELECT line" stating the variable(s) to be unified and a SPARQL body following the keyword WHERE, denoted by $\mathtt{body}(\mathtt{Q})$ in this paper. The SPARQL body is a sequence of triples and FILTER assertions separated by dots which mean "and". Given an RDFS base $\mathcal{B}$ and a SPARQL query $\mathtt{Q}$, the execution of $\mathtt{Q}$ on $\mathcal{B}$ gives a set of bindings of the variables of $\mathtt{Q}$'s SELECT line corresponding to matchings of the $\mathtt{Q}$'s body with $\mathcal{B}$. This set is denoted by $\mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B})$. For example, if $\mathtt{Q}$ is the query of Figure 1, if $\mathcal{B}$ contains the recipe description of the tarte Tatin (containing apples and with a preparation time of 65 minutes) and a cooking ontology stating that $\mathcal{B} \vdash \langle \mathtt{Apple\ subc\ Fruit} \rangle$, then $\mathtt{exec}_{\vdash}(\mathtt{Q}, \mathcal{B})$ contains the binding pair $(\mathtt{?r}, \mathtt{tarteTatin})$. [3]

Remark about the notation. An RDFS triple, such as $\langle \mathtt{?x\ type\ Fruit} \rangle$ appears with a slightly different syntax as an assertion of a SPARQL query body. However, in this paper, we will use the two notations interchangeably. Of course, this involves the use of translation procedures in the code.

3 Three application problems

3.1 Recipe retrieval in the CBR system TAAABLE

TAAABLE [7] is a CBR system that was originally developed as a contestant of the CCC (the computer cooking contest, organized within most ICCBR conferences since 2008). A contestant of the CCC aims at answering cooking queries such as

$$\mathcal{Q} = \text{"I want a dessert recipe with pears and butter but without cinnamon."} \quad (1)$$

For this purpose, it reuses a recipe base provided by the contest. To this end, TAAABLE first searches for a recipe from the base that exactly matches $\mathcal{Q}$. If no such recipe exists, TAAABLE minimally modifies $\mathcal{Q}$ into $\mathcal{Q}'$ so that there exists at least one recipe exactly matching $\mathcal{Q}'$. For example, if $\mathcal{Q}' = \sigma(\mathcal{Q})$ with $\sigma = \mathtt{pear} \rightsquigarrow \mathtt{fruit} \circ \mathtt{butter} \rightsquigarrow \mathtt{margarine}$ (i.e., σ is the substitution of $\mathtt{pear}$ by $\mathtt{fruit}$ and of $\mathtt{butter}$ by $\mathtt{margarine}$), an apple crumble with margarine and without cinnamon can be selected in the recipe

[3] This presentation of RDFS and SPARQL is simplified to fit the needs of this paper. : needed many features are not considered here, but they are not used in this paper.

base (because the piece of knowledge "apples are fruit" is in the TAAABLE domain ontology), and then adapted to answer $\mathcal{Q}$ by substituting apples with pears and margarine with butter. Other adaptation issues (of ingredient quantity and of the preparation) are also studied in TAAABLE but not considered here.

A semantic wiki called WIKITAAABLE [8] (`wikitaaable.loria.fr`) has been developed in order to manage the TAAABLE knowledge base which contains its domain ontology (hierarchies of ingredients, etc.) and the recipe base. WIKITAAABLE is implemented thanks to the semantic wiki engine SMW (`www.semantic-mediawiki.org`) that comes with an RDFS export, hence the idea of using RDFS techniques and tools for TAAABLE. This idea has led to TUUURBINE, a CBR retrieval engine using SPARQL [11]. More precisely, a TUUURBINE query combines SPARQL queries. For example, the query $\mathcal{Q}$ of equation (1) is based on the following SPARQL queries:

$$\mathtt{Q}_+ = \boxed{\begin{array}{l}\text{SELECT ?r}\\ \text{WHERE \{?r type DessertRecipe .}\\ \qquad\quad \text{?r ing ?x . ?x type Pear .}\\ \qquad\quad \text{?r ing ?y . ?y type Butter\}}\end{array}}$$

$$\mathtt{Q}_- = \boxed{\begin{array}{l}\text{SELECT ?r}\\ \text{WHERE \{?r ing ?x . ?x type Cinnamon\}}\end{array}}$$

The recipes matching $\mathcal{Q}$ are in the set $\mathtt{exec}_{\vdash}(\mathtt{Q}_+, \mathcal{B}) \setminus \mathtt{exec}_{\vdash}(\mathtt{Q}_-, \mathcal{B})$ where $\mathcal{B}$ is the base exported from WIKITAAABLE: this gives the recipes matching $\mathtt{Q}_+$ and not $\mathtt{Q}_-$. $\mathcal{B}$ contains the recipe and the ontology. This latter contains the fact that apples and pears are fruits: $\mathcal{B} \vdash \{\langle \mathtt{Apple\ subc\ Fruit}\rangle, \langle \mathtt{Pear\ subc\ Fruit}\rangle\}$. The modification of $\mathcal{Q}$ into $\mathcal{Q}'$ consists in a modification of $\mathtt{Q}_+$ into $\mathtt{Q}'_+$ (keeping $\mathtt{Q}_-$ unchanged, i.e., $\mathtt{Q}'_- = \mathtt{Q}_-$):

$$\mathtt{Q}'_+ = \boxed{\begin{array}{l}\text{SELECT ?r}\\ \text{WHERE \{?r type DessertRecipe .}\\ \qquad\quad \text{?r ing ?x . ?x type Fruit .}\\ \qquad\quad \text{?r ing ?y . ?y type Margarine\}}\end{array}}$$

Therefore, a first module for transforming SPARQL queries has been developed for TUUURBINE. This module, though it has a general purpose, remained limited and proved to be insufficient for the application described hereafter.

3.2 Approximate search in the corpus of Henri Poincaré letters

The famous mathematician Henri Poincaré has had a long correspondence with many people, including scientists of his time. The letters he has written and received are gathered in the "HP papers corpus" (`henripoincarepapers.univ-lorraine.fr`), which has been scanned and indexed. Currently, this index is being migrated into RDFS annotations. For example, the following triples concern the letter number 12 that has been sent by H. Poincaré to David Hilbert and that is about hyperbolic geometry:

$$\langle \mathtt{letter12\ isSentBy\ hPoincaré}\rangle\langle \mathtt{letter12\ isSentTo\ dHilbert}\rangle$$
$$\langle \mathtt{letter12\ hasForTopic\ ?t}\rangle\langle \mathtt{?t\ type\ HyperbolicGeometry}\rangle$$

The RDFS base $\mathcal{B}$ of the H. Poincaré corpus contains such annotations about letters as well as information about some persons and organizations (e.g., $\mathcal{B} \vdash \langle \texttt{dHilbert type Mathematician} \rangle$), and an ontology related to the domain (e.g., $\mathcal{B} \vdash \langle \texttt{Mathematician subc Scientist} \rangle$).

Therefore, the letters of this corpus sent to a geometer before 1895 are in $\texttt{exec}_{\vdash}(\texttt{Q}, \mathcal{B})$ with $\texttt{Q} =$

```
SELECT ?ℓ
WHERE {?ℓ isSentTo ?x . ?x type Geometer .
       ?ℓ dateOfExpedition ?d .
       FILTER (?d< '01/01/1895')}
```

.

Now, it happens that an exact search is not always sufficient. Such situations are described in [4]. Two examples are given below.

First, consider $\texttt{Q} =$

```
SELECT ?ℓ
WHERE {?ℓ isSentBy hPoincaré .
       ?ℓ isSentTo gMittagLeffler}
```

. If $\texttt{exec}_{\vdash}(\texttt{Q}, \mathcal{B}) = \emptyset$, that does not mean that no letters has been written by H. Poincaré to Gösta Mittag-Leffler. It could mean that such a letter has been written but was lost. Now, consider the query $\texttt{Q}' =$

```
SELECT ?ℓ
WHERE {?ℓ isSentTo hPoincaré .
       ?ℓ isSentBy gMittagLeffler}
```

. $\texttt{Q}'$ is obtained by exchanging sender and recipient in $\texttt{Q}$. The query transformation $\texttt{Q} \mapsto \texttt{Q}'$ is relevant for historians since, when searching a letter from H. Poincaré to G. Mittag-Leffler, accessing the letters from the latter to the former can be useful, because such a letter can be a response to a letter that has disappeared.

Now, consider the query for letters sent by D. Hilbert at "the end of the XIXth century". This period of time is imprecisely specified, so an interval of time is chosen to model it, e.g., $[1890, 1900]$, hence the query $\texttt{Q} =$

```
SELECT ?ℓ
WHERE {?ℓ isSentBy dHilbert . ?ℓ dateOfExpedition ?d .
       FILTER (?d>= '01/01/1890') . FILTER (?d<= '31/12/1900')}
```

.

Now, a letter of David Hilbert of 1887 or even 1902 would be an acceptable answer to the informal query, whereas it does not answer the formal query $\texttt{Q}$. Hence the usefulness of transformations $\texttt{Q} \mapsto \texttt{Q}'$ and $\texttt{Q} \mapsto \texttt{Q}''$ corresponding to the enlargement of the interval of time to $[1885, 1900]$ and $[1890, 1905]$, respectively. $\texttt{Q}'$ (resp., $\texttt{Q}''$) is obtained by substituting `'01/01/1890'` (resp., `'31/12/1900'`) with `'01/01/1885'` (resp., `'31/12/1905'`) in $\texttt{Q}$.

Other examples have been considered. Some of them consist in generalizing classes in the query (as for the `Pear` to `Fruit` example in Section 3.1). Another one consists in replacing a person by another one that is close in a relationship network.

From this study and the previous one has emerged the need to develop a generic tool for managing SPARQL query transformations. A rule language for this purpose—SQTRL—has been developed as well as a system for managing such rules (this is detailed in Section 4). The principle of approximate search in the H. Poincaré letters using SQTRL rules is the one of a search in a state space where a state is a SPARQL

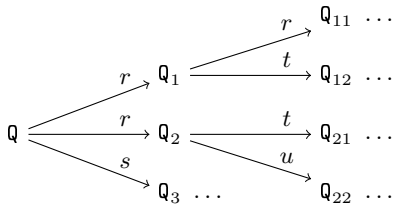

Fig. 2. A SPARQL query state space. $\mathtt{Q}$ is the initial query. r, s, t and u are SQTRL rules.

query, the initial state is the initial query, and transitions correspond to rule applications, as illustrated in Figure 2. Such a space is searched by increasing "transformation costs" using e.g., dynamic programming. These costs are associated to rules and are assumed to be additive. For example, the transformation from $\mathtt{Q}$ to $\mathtt{Q}_{21}$ in Figure 2 is $\mathtt{cost}(r) + \mathtt{cost}(t)$. This principle is also applied for retrieval in TAAABLE.

3.3 Cocktail name adaptation in the CBR system TAAABLE

TAAABLE has been applied for cocktail recipes for the CCCs of 2014 and 2015. For example, given the query $\mathcal{Q}$ = "I want a cocktail with schnapps and hot chocolate", TUUURBINE, the retrieval engine of TAAABLE, finds a recipe R named `"Irish coffee"` and the adaptation consists in applying the substitution $\sigma = \mathtt{whisky} \rightsquigarrow \mathtt{schnapps} \circ \mathtt{coffee} \rightsquigarrow \mathtt{hotChocolate}$ on the ingredients of R. The adapted recipe is denoted by $\sigma(R)$. The cocktail name adaptation problem is how to name the adapted cocktail recipe $\sigma(R)$, given the name of R (the string `"Irish coffee"`) and the substitution σ. This issue has been addressed in [12]. It is based on an RDFS base β_R associated with the recipe R and giving a (partial or complete) explanation of its name. In the example:

$$\begin{aligned}\beta_R = \{&\langle \mathtt{coffee\ englishName\ "coffee"}\rangle, \langle \mathtt{?nameR\ superStringOf\ "coffee"}\rangle,\\ &\langle \mathtt{whisky\ hasOrigin\ ireland}\rangle, \langle \mathtt{ireland\ hasEnglishAdjective\ "Irish"}\rangle,\\ &\langle \mathtt{?nameR\ superStringOf\ "Irish"}\rangle\}\end{aligned}$$

where `coffee` and `whisky` are some R ingredient types and `?nameR` is the variable that is associated with the recipe name. Let $\beta_{\sigma(R)}$ be the RDFS base obtained by applying the substitution σ on β_R:

$$\begin{aligned}\beta_{\sigma(R)} = \{&\langle \mathtt{hotChocolate\ englishName\ "coffee"}\rangle, \langle \mathtt{?nameR\ superStringOf\ "coffee"}\rangle,\\ &\langle \mathtt{schnapps\ hasOrigin\ ireland}\rangle, \langle \mathtt{ireland\ hasEnglishAdjective\ "Irish"}\rangle,\\ &\langle \mathtt{?nameR\ superStringOf\ "Irish"}\rangle\}\end{aligned}$$

$\beta_{\sigma(R)}$ does not match the domain knowledge in the sense that $\mathtt{exec}_{\vdash}(\mathtt{Q}_{\sigma(R)}, \mathcal{B}) = \emptyset$, with

$\mathtt{Q}_{\sigma(R)} =$

```
SELECT ?anyVariable
WHERE {hotChocolate  englishName  "coffee"   .
       schnapps  hasOrigin  ireland   .
       ireland  hasEnglishAdjective  "Irish"}
```

(?anyVariable being a variable that is used only for syntax purpose). The transformation here consists in substituting identified resources or literals with variables, knowing that the resources occurring in $\sigma(R)$ (i.e., hotChocolate and schnapps). In the example, the substitutions "coffee" $\rightsquigarrow$?x, ireland $\rightsquigarrow$?y and "Irish" $\rightsquigarrow$?z can be done (among others: one could have also substituted a property by a variable). A final state of this search is a query $\mathtt{Q}_{\mathtt{gen}}$ such that $\mathtt{exec}_{\vdash}(\mathtt{Q}_{\mathtt{gen}}, \mathcal{B}) \neq \emptyset$, for example:

$\mathtt{Q}_{\mathtt{gen}} =$

```
SELECT ?anyVariable
WHERE {hotChocolate  englishName  ?x  .
       schnapps  hasOrigin  ?y  .
       ?y  hasEnglishAdjective  ?z}
```

Here, it is assumed that the execution of this query gives exactly one binding:

$$\mathtt{exec}_{\vdash}(\mathtt{Q}_{\mathtt{gen}}, \mathcal{B}) = \{\{\mathtt{?x}, \mathtt{"hot\ chocolate"}\}, \{(\mathtt{?y}, \mathtt{germany}\}, \{(\mathtt{?z}, \mathtt{"German"})\}\}$$

Composing the generalizations done from $\beta_{\sigma(R)}$ to $\beta_{\mathtt{gen}}$ and this binding, it comes the substitutions $\sigma_1 =$ "coffee" $\rightsquigarrow$ "hot chocolate" and $\sigma_2 =$ "Irish" $\rightsquigarrow$ "German". Since the literals "coffee" and "hot chocolate" are linked in β_R with ?nameR, the adaptation consists in applying these substitutions, hence the proposed name of the adapted recipe:

$$\sigma_1(\sigma_2(\mathtt{"Irish\ coffee"})) = \mathtt{"German\ hot\ chocolate"}$$

4 SQTRL: a language for SPARQL query transformation rules

The language SQTRL presented below has emerged from the need to transform SPARQL queries as presented above. After the presentation of this language, examples of SQTRL rules that cover the examples of Section 3 are given. Finally, the system that manages these rules is briefly described.

4.1 SQTRL: syntax and application of the rules

A SQTRL rule is defined in an XML syntax as follows (the texts in italics have to be substituted by the appropriate strings):

```
<rule name=name of the rule>
  <context>RDFS triples under the SPARQL syntax</context>
  <left>RDFS triples under the SPARQL syntax</left>
  <right>RDFS triples under the SPARQL syntax</right>
  <cost>a float</cost>
  <explanation>a text possibly using variables</explanation>
</rule>
```

If r is such a rule, the contents in the fields with tags <context>, <left>, <right> and <cost> are denoted by $\mathtt{context}(r)$, $\mathtt{left}(r)$, $\mathtt{right}(r)$ and $\mathtt{cost}(r)$.

For example, the following rule substitutes a class C by a class D provided that C occurs as an object in a triple of the query Q body and that C is a subclass of D:

$r =$
```
<rule name=Generalize an object class>
  <context>?C  subc  ?D</context>
  <left>?x  ?p  ?C</left>
  <right>?x  ?p  ?D</right>
  <cost>1.0</cost>
  <explanation>Generalize ?C in ?D</explanation>
</rule>
```
(2)

Thus, $\texttt{context}(r) =$ `?C subc ?D`, $\texttt{left}(r) =$ `?x ?p ?C`, $\texttt{right}(r) =$ `?x ?p ?D`, and $\texttt{cost}(r) = 1$.

Let Q be a SPARQL query and $\mathcal{B}$ be an RDFS base representing relevant domain knowledge. For the example, Q =
```
SELECT ?r
WHERE {?r  type  TartDishRecipe  .
       ?r  ing  ?i  .  ?i  type  Pear}
```
and $\mathcal{B}$ such that $\mathcal{B} \vdash \{\langle \texttt{Pear subc Fruit}\rangle, \langle \texttt{TartDish subc DishWithPastry}\rangle\}$. Let r be a SQTRL rule, for the example, it is the rule defined in equation (2). The application of r to Q given $\mathcal{B}$ is the set $\texttt{apply}(r, \texttt{Q}, \mathcal{B})$ of the queries Q′ obtained by transforming Q using rule r, knowing $\mathcal{B}$. If $\texttt{apply}(r, \texttt{Q}, \mathcal{B}) = \emptyset$, the rule r is said to be non applicable on Q given $\mathcal{B}$.

Figure 3 presents the algorithm for computing $\texttt{apply}(r, \texttt{Q}, \mathcal{B})$ and illustrates the algorithm with the example values r, Q and $\mathcal{B}$ given above.

Note about the computing time. Applying an SQTRL rule amounts mainly to execute a few SPARQL queries. Now, executing a SPARQL query amounts mainly is finding subgraph isomorphisms (between the body of the query to the graph representing the RDFS base), which is known to be an NP-complete problem. However, one must keep in mind that the size of the parameter to be taken into account is mainly the size of the query, which, for rule application, corresponds to the size of the context and of the left part of the rule. Thus, unless the rule is huge—a situation that has not occurred in our applications—the application of a rule is very quick, especially thanks to the use of an efficient SPARQL execution tool. So, our intuition is that the application of SQTRL rules is fast in practice though an accurate complexity estimation as well as some experiments remain to be done.

The same argument can be given for case retrieval and case adaptation: though they are based on search in a state space, if the size of the initial query is reasonable, then the search in this space does not take too much time in practice, and that is what we have experimented with TAAABLE. If the closest case is very dissimilar to the query, this involves a result that is likely to be of poor quality: for example, if the user queries TAAABLE for a vegetarian recipe with pastry and pineapple and the only recipe in the case base is for beef Stroganov, then it is unlikely that the users will be satisfied with the adaptation query (despite the adaptation capabilities of TAAABLE). For this

function apply(r : *SQTRL rule*, Q : *SPARQL query*, $\mathcal{B}$: *RDFS base*)
begin
  ▷ *Initialize* producedQueries*, the output of the algorithm*
  producedQueries ← ∅
  ▷ *Unification of the context of* r *with* $\mathcal{B}$
① ctxtToBase ← exec$_\vdash$ (SELECT $\mathcal{V}$(context(r)) WHERE {context(r)}, $\mathcal{B}$)
  for binding ∈ ctxtToBase **do**
② boundLeft ← application of binding to left(r)
③ boundRight ← application of binding to right(r)
    ▷ *Unification of the bound left part of* r *to the body of* Q
④ leftToQuery ← exec$_\vdash$ (SELECT $\mathcal{V}$(boundLeft) WHERE {boundLeft}, body(Q))
    for binding′ ∈ leftToQuery **do**
⑤ boundLeft′ ← application of binding′ to boundLeft
⑥ boundRight′ ← application of binding′ to boundRight
      Q′ ← copy of Q
      Remove the triples of boundLeft′ from the body of Q′
⑦ Add the triples of boundRight′ to the body of Q′
      producedQueries ← producedQueries ∪ {Q′}
    end
  end
  return producedQueries
end

① ctxtToBase = exec$_\vdash$ (SELECT ?C ?D WHERE {?C subc ?D}, $\mathcal{B}$) = {binding$_1$, binding$_2$}
with binding$_1$ = {(?C, Pear), (?D, Fruit)}
and binding$_2$ = {(?C, TartDishRecipe), (?D, DishWithPastryRecipe)}.
The following of the explanation is with binding = binding$_1$.

② Since left(r) = ?x ?p ?C, boundLeft = ?x ?p Pear.

③ Since right(r) = ?x ?p ?D, boundRight = ?x ?p Fruit.

④ leftToQuery = exec$_\vdash$ (SELECT ?x ?p WHERE {?x ?p Pear}, body(Q)) which gives the set {binding′} (only one binding in this example) with binding′ = {(?x, ?i), (?p, type)}.

⑤ boundLeft′ = ?i type Pear.

⑥ boundRight′ = ?i type Fruit.

⑦ Q′ =
```
SELECT ?r
WHERE {?r  type  TartDishRecipe  .
       ?r  ing  ?i  .  ?i  type  Fruit}
```

Fig. 3. The SQTRL rule application: the algorithm and an example ($\mathcal{V}(s)$ denotes the set of variables occurring in a sequence s of RDFS triples).

reason, a timeout interruption is used: after a too long computing time for retrieval, it is considered that the adaptation procedure will not be able to make enough modifications to achieve a satisfying result.

4.2 Examples of SQTRL rules

This section presents some SQTRL rules that cover the examples presented in Section 3.

Generalization rules. Let $\mathtt{Q}_1$ and $\mathtt{Q}_2$ be two SPARQL queries with the same SELECT line. $\mathtt{Q}_1$ is said to be less general than $\mathtt{Q}_2$—denoted by $\mathtt{Q}_1 \sqsubseteq \mathtt{Q}_2$—if for every RDFS base $\mathcal{B}$, $\mathtt{exec}_{\vdash}(\mathtt{Q}_1, \mathcal{B}) \subseteq \mathtt{exec}_{\vdash}(\mathtt{Q}_2, \mathcal{B})$. A generalization rule is a rule r such that for every SPARQL query $\mathtt{Q}$, every RDFS base $\mathcal{B}$ and every $\mathtt{Q}' \in \mathtt{apply}(r, \mathtt{Q}, \mathcal{B})$, $\mathtt{Q} \sqsubseteq \mathtt{Q}'$. The rule r of equation (2) is such a rule, as way as the rule `"Generalize a subject class"` obtained by replacing $\mathtt{left}(r)$ by `?C ?p ?x` and $\mathtt{right}(r)$ by `?D ?p ?x`. Similarly, the following rule is a rule that generalizes predicates:

```
<rule name=Generalize a property in predicate position>
  <context>?p  subp  ?q</context>
  <left>?x  ?p  ?y</left>
  <right>?x  ?q  ?y</right>
  <cost>1.0</cost>
  <explanation>Generalize ?p in ?q</explanation>
</rule>
```

A second way to generalize a query is by removing a triple:

```
<rule name=Remove a triple from the body of the query>
  <context></context>
  <left>?x  ?p  ?y</left>
  <right></right>
  <cost>1.0</cost>
  <explanation>Remove the triple ?x  ?p  ?y</explanation>
</rule>
```

It can be noted that removing a triple can "disconnect" variables that were previously connected by a path. For example, if
$\mathtt{body}(\mathtt{Q}) =$ `s p ?x . ?x ?q ?y . ?y ?r ?z` then the removal of `?x ?q ?y` disconnects `?z` from `s`. A variant of the above rule exists that prevent such situation.

A third type of generalization rules works with filters, when a filter has the form FILTER `(?x` $\bowtie$ $v)$ where $\bowtie \in \{<, <=, >=, >\}$ and v is a value of a numerical type (or a date). It consists in replacing v by $v + c$ where c is some numerical constant such that $c > 0$ if $\bowtie \in \{<, <=\}$ and $c < 0$ else. Such a rule does not follow the syntax of other rules (it has a specific syntax), and has been motivated by the modeling of "the end of the XIXth century" issue (cf. Section 3.2).

A final type of generalization rules, that has been used in Section 3.3, consists in substituting a constant c (i.e., a resource or a literal) by a variable ?c:

```
<rule name=Generalize subject c by ?c>
  <context></context>
  <left>c  ?p  ?x</left>
  <right>?c  ?p  ?x</right>
  <cost>1.0</cost>
  <explanation>Generalization of c in ?c</explanation>
</rule>
```

Similar rules are defined for substituting a constant by a variable in predicate and object positions. Using such rules can be used to transform $\mathtt{Q}_{\sigma(R)}$ into $\mathtt{Q}_{\mathtt{gen}}$ (cf. Section 3.3).

These generalization rules can be qualified as application-independent as they can be used in various applications. By contrast, some other SQTRL rules are strongly related to applications.

Application-dependent rules. The generalized rules presented above cover some of the examples presented in Section 3 but not all of them. For example, in Section 3.1, it is said that, for a dessert recipe, butter can be replaced by margarine and vice-versa, which can be formalized thanks to two rules, the first one being

```
<rule name=Replace butter by margarine in a dessert>
  <context>?r  type  DessertRecipe</context>
  <left>?r  ing  ?x  .  ?x  type  Butter</left>
  <right>?r  ing  ?x  .  ?x  type  Margarine</right>
  <cost>0.1</cost>
  <explanation>Replace butter with margarine</explanation>
</rule>
```

and the other one being a similar rule obtained by exchanging the terms Butter and Margarine in the first rule. This kind of rules can also be used as adaptation rules: given a dessert recipe R with butter and a query of a dessert recipe with margarine, the above rule can be applied to adapt R to answer the query. Thus, SQTRL can also be used as a language for adaptation rules. In a way, it could be said that retrieval *adapts* the query to fit at least one case from the case base that is then adapted to fit the query.

Another domain-dependent rule is the one exchanging sender and recipient in H. Poincaré letters, which can be formalized by:

```
<rule name=Exchange sender and recipient>
  <context></context>
  <left>?x  isSentTo  ?y  .  ?x  isSentBy  ?z</left>
  <right>?x  isSentBy  ?y  .  ?x  isSentTo  ?z</right>
  <cost>1.0</cost>
  <explanation>Exchange sender/recipient: ?y/?z</explanation>
</rule>
```

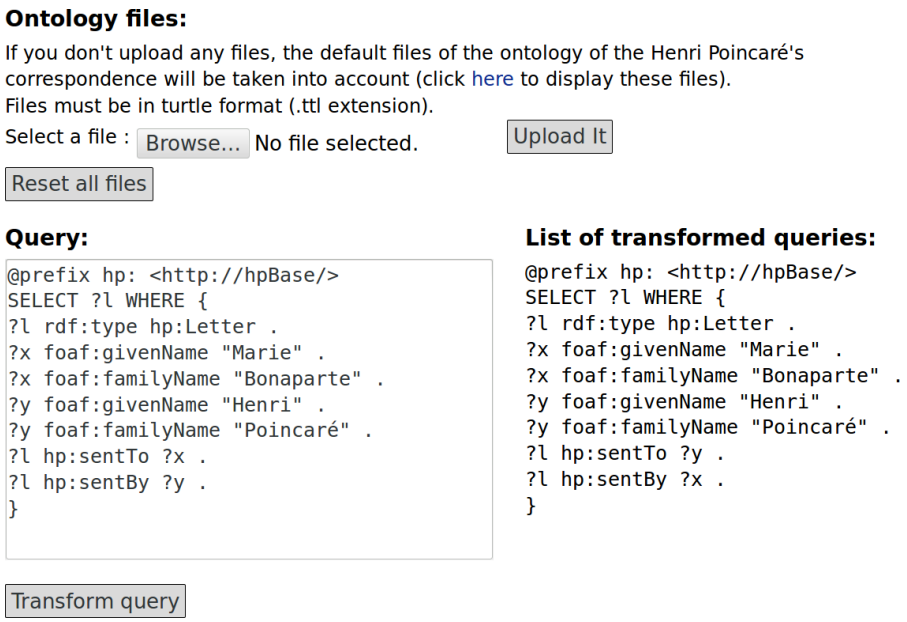

Fig. 4. A screenshot of the SQTRL tool demo. The query on the left has been transformed in the list with only one query on the right, the applied rule being the one named `"Exchange sender and recipient"`.

4.3 A tool for managing SQTRL rules

A tool has been developed to manage SQTRL rules with the following functionalities: creation, serialization and application of a rule. It uses the rule syntax presented above and the turtle syntax for RDFS base. It uses the RDFS and SPARQL management tool KGRAM [6] and is written in Java. It is freely accessible at `http://tuuurbine.loria.fr/sqtrl/` (a site with the code and a user manual). A demo has been developed as illustrated by the screenshot of Figure 4.

5 Discussion and related work

Case retrieval based on minimal generalization of the query is not a new idea in CBR. For example, it was applied with SQL queries for a translation system based on CBR in the early 1990s [20]. It was also applied in a graph formalism for representing molecular structures a few years later [14]. TAAABLE has been using this principle since its first version, though the use of RDFS and SPARQL has only been developed since 2014. The originality of this work is that it presents a well-defined rule language for transforming queries and that it is based on RDFS and SPARQL that are semantic web standards that are getting more and more used and with which data and knowledge are represented and accessed freely within the Linked Open Data [2].

Case adaptation based on minimal modification of the source case is not a new idea either. Actually, when the modification is based on generalizations, it is related to generalization methods found in the early years of machine learning [10] and applied to CBR [19]. Here SQTRL rules are used to implement this idea (for both generalization-based and non generalization-based modifications) when (a part of) the source case can be represented as an RDFS base or a SPARQL query. Another work based on this principle is revision-based adaptation [5], i.e., adaptation based on the use of a belief revision operator [1]. Such an operator $\dot{+}$ associates to two belief bases ψ and μ a belief base $\psi \dot{+} \mu$ equivalent to $\psi' \wedge \mu$ where ψ' is the minimal modification of ψ to make it consistent with μ (given some modification metric). Therefore, the idea of $\dot{+}$-adaptation is to make the revision of the source case by the query (knowing that both have to be consistent with the domain knowledge). The difference here is that RDFS bases are hardly inconsistent.[4] Therefore, belief revision of RDFS bases in the classical sense has little interest. Currently, alternative ways of defining revision in RDFS are investigated which could lead to a unification of the approach presented here with revision-based adaptation.

This work has strong connections with the theory developed last years that is based on amalgams and on refinement operators [17, 16]. Indeed, refinement operators can be likened to SQTRL rules and are used both for case retrieval [16] and for (single and multiple) case adaptation [17]. Two differences between this previous work and the current one can be pointed out. First, the amalgalms and refinement theory is defined independently from a knowledge representation formalism, whereas we present an approach more concrete, with the advantage of being associated with an operational tool. Second, the refinement operators are generalization and specialization operators, whereas SQTRL allows to define non generalization rules. One could argue that such a rule can be "simulated" by the application of two rules, one for generalization and one for specialization. For example, the rule making the substitution `Butter` $\rightsquigarrow$ `Margarine` can be seen as the composition of the generalization $g =$ `Butter` $\rightsquigarrow$ `Fat` and the specialization $s =$ `Fat` $\rightsquigarrow$ `Margarine`. However, applying these two rules has a cost $\texttt{cost}(g) + \texttt{cost}(s)$ that may be too high, if the `Fat` class contains subclasses less close to `Butter` and `Margarine` from a cooking viewpoint, such as `DuckFat`.[5] Therefore a rule substituting "directly" butter by margarine with a lower cost is useful and so are other non generalization rules.

6 Conclusion and future work

This paper has presented SQTRL, a query transformation rule language and tool adapted to the SPARQL formalism, and three of its applications. Our claim is that this language and this tool can be applied in many application domains of CBR, provided that the

[4] The only case of inconsistency of an RDFS base is related to a type error property for datatype properties. For example, if the age of an individual stated with property `age` is an integer, then the triple $\langle$`juliet age true`$\rangle$ is inconsistent. Such situations of inconsistencies are not relevant here.

[5] These classes are taken from WIKITAAABLE, the semantic wiki that contains TAAABLE ontology: `http://wikitaaable.loria.fr`.

case language and the domain knowledge can be translated into RDFS, which embeds the feature-value formalisms and taxonomy languages frequently used in CBR [13]. In particular, it is planned to use SQTRL for an ongoing work in a medical domain.

The SQTRL tool is in the TUUURBINE web site, but the current version of TUUURBINE does not use it: the integration of these tools is a future work.

SQTRL is in its first stable version but, surely, this language will require evolutions. Our policy is to make it evolve when new needs emerge. The particular treatment associated with the filters show that there is room for improvement here. The difficulty is that, in general, the filter term can use the Boolean operators (`and`, `or`, `not`), which raises specific issues if the goal is to make transformations up to equivalence (and not only syntactical ones): if $\mathtt{Q}_1$ and $\mathtt{Q}_2$ are equivalent queries (that is $\mathtt{Q}_1 \sqsubseteq \mathtt{Q}_2$ and $\mathtt{Q}_2 \sqsubseteq \mathtt{Q}_1$) and r is a rule that can be applied to $\mathtt{Q}_1$ to give birth to $\mathtt{Q}'_1$, then r should be applicable to $\mathtt{Q}_2$ to give birth to a query $\mathtt{Q}'_2$ equivalent to $\mathtt{Q}'_1$.

In this paper, the cost fields of the rules are given arbitrarily. One question is how to fix them, which is a complex knowledge acquisition issue often met in CBR (similar to the choice of weights in a similarity measure). Another point is that, in this first version of SQTRL, costs are constant, whereas one can consider that they should be parameterized by the bindings. For example, using the generalization rule of equation (2), one can argue that it is less costly to make the generalization `WilliamsPear` $\rightsquigarrow$ `Pear` than the generalization `Apple` $\rightsquigarrow$ `Fruit` in the cooking domain. Therefore, having a cost field that is linked with a function will probably be useful.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments that have helped to improve this paper.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the Logic of Theory Change: partial meet functions for contraction and revision. Journal of Symbolic Logic **50** (1985) 510–530
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data – the story so far. Semantic services, interoperability and web applications: emerging concepts (2009) 205–227
3. Brickley, D., Guha, R.V.: RDF Schema 1.1, `https://www.w3.org/TR/rdf-schema/`, W3C recommendation, last consultation: March 2017 (2014)
4. Bruneau, O., Garlatti, S., Guedj, M., Laubé, S., Lieber, J.: SemanticHPST: Applying Semantic Web Principles and Technologies to the History and Philosophy of Science and Technology. In Gandon, F., Zimmermann, A., Faron-Zucker, C., Breslin, J., Villata, S., Guéret, C., eds.: The Semantic Web: ESWC 2015 Satellite Events . Volume 9341 of Lecture Notes in Computer Science., Portoroz, Slovenia, Springer International Publishing (May 2015) 416–427
5. Cojan, J., Lieber, J.: Applying Belief Revision to Case-Based Reasoning. In Prade, H., Richard, G., eds.: Computational Approaches to Analogical Reasoning: Current Trends. Volume 548 of Studies in Computational Intelligence. Springer (2014) 133 – 161
6. Corby, O., Gaignard, A., Faron-Zucker, C., Montagnat, J.: KGRAM Versatile Data Graphs Querying and Inference Engine. In: Proc. IEEE/WIC/ACM International Conference on Web Intelligence, Macau (December 2012)

7. Cordier, A., Dufour-Lussier, V., Lieber, J., Nauer, E., Badra, F., Cojan, J., Gaillard, E., Infante-Blanco, L., Molli, P., Napoli, A., Skaf-Molli, H.: Taaable: a Case-Based System for personalized Cooking. In Montani, S., Jain, L.C., eds.: Successful Case-based Reasoning Applications-2. Volume 494 of Studies in Computational Intelligence. Springer (2014) 121–162
8. Cordier, A., Lieber, J., Molli, P., Nauer, E., Skaf-Molli, H., Toussaint, Y.: WikiTaaable: A semantic wiki as a blackboard for a textual case-based reasoning system. In: SemWiki 2009 – 4th Semantic Wiki Workshop, Heraklion, Greece (May 2009)
9. Cox, M.T., Muñoz-Avila, H., Bergmann, R.: Case-based planning. Knowledge Engineering Review **20**(3) (2005) 283–287
10. Dietterich, T.G., Michalski, R.S.: A Comparative Review of Selected Methods for Learning from Examples. In: Machine Learning. Springer-Verlag (1983) 41–81
11. Gaillard, E., Infante-Blanco, L., Lieber, J., Nauer, E.: Tuuurbine: A Generic CBR Engine over RDFS. In: Case-Based Reasoning Research and Development. Volume 8765., Cork, Ireland (September 2014) 140 – 154
12. Kiani, N., Lieber, J., Nauer, E., Schneider, J.: Analogical Transfer in RDFS, Application to Cocktail Name Adaptation. In Goel, A., Diaz-Agudo, M.B., Roth-Berghofer, T., eds.: International Conference on Case-Based Reasoning (ICCBR-2016). Volume 9969 of Case-Based Reasoning Research and Development, 24th International Conference, ICCBR 2016., Atlanta, United States, Springer (October 2016) 218 – 233
13. Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann, Inc. (1993)
14. Lieber, J., Napoli, A.: Using Classification in Case-Based Planning. In Wahlster, W., ed.: Proceedings of the 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Hungary, John Wiley & Sons, Ltd. (1996) 132–136
15. Minor, M., Bergmann, R., Görg, S.: Case-based adaptation of workflows. Information Systems **40** (2014) 142–152
16. Ontañón, S., Shokoufandeh, A.: Refinement-Based Similarity Measures for Directed Labeled Graphs. In: International Conference on Case-Based Reasoning, Springer (2016) 311–326
17. Ontañón, S., Plaza, E.: Amalgams: A formal approach for combining multiple case solutions. In: International Conference on Case-Based Reasoning, Springer (2010) 257–271
18. Recio-García, J.A., Sánchez, A., Díaz-Agudo, B., González-Calero, P.A.: JColibri 1.0 in a nutshell. A software tool for designing CBR systems. In M. Petridis, e., ed.: Proccedings of the 10th UK Workshop on Case Based Reasoning, CMS Press, University of Greenwich (2005) 20–28
19. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey (1989)
20. Shimazu, H., Kitano, H., Shibata, A.: Retrieving Cases from Relational Data-Bases: Another Stride Towards Corporate-Wide Case-Based Systems. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), Chambéry. (1993) 909–914
21. Watson, I.: Is CBR a Technology or a Methodology? In del Pobil, A.P., Mira, J., Alis, M., eds.: Tasks and Methods in Applied Artificial Intelligence – Volume II of the Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Experts Systems. LNCS 1416, Springer (1998) 525–534